# Solving Bin Packing Problems Using VRPSolver Models

Artur Alves Pessoa, Ruslan Sadykov, Eduardo Uchoa

# Solving Bin Packing Problems Using VRPSolver Models

Artur Pessoa[2], Ruslan Sadykov[1], and Eduardo Uchoa[2]

[1]INRIA Bordeaux – Sud-Ouest 200 Avenue de la Veille Tour, 33405 Talence, France
[2]Universidade Federal Fluminense - Engenharia de Produção , Rua Passo da Pátria 156, Niterói - RJ - Brasil - 24210-240,

3 November 2020

**Abstract**

We propose branch-cut-and-price algorithms for the classic bin packing problem and also for the following related problems: vector packing, variable sized bin packing and variable sized bin packing with optional items. The algorithms are defined as models for VRPSolver, a generic solver for vehicle routing problems. In that way, a simple parameterization enables the use of several branch-cut-and-price advanced elements: automatic stabilization by smoothing, limited-memory rank-1 cuts, enumeration, hierarchical strong branching and limited discrepancy search diving heuristics. As an original theoretical contribution, we prove that the branching over accumulated resource consumption (Gélinas et al. 1995), that does not increase the difficulty of the pricing subproblem, is sufficient for those bin packing models. Extensive computational results on instances from the literature show that the VRPSolver models have a performance that is very robust over all those problems, being often superior to the existing exact algorithms on the hardest instances. Several instances could be solved to optimality for the first time.

## 1 Introduction

Bin packing problems are among the most classic combinatorial optimization problems, being discussed since the 1930s [22]. The problems addressed in this paper can be defined as follows:

- **Bin Packing Problem (BPP).** Let $I = \{1, \ldots, \mathcal{I}\}$ be a set of $\mathcal{I}$ items and assume an unlimited quantity of identical bins with integer positive capacity $Q$. Each $i \in I$ has integer positive weight $w_i \leq Q$. The goal is finding a packing using the minimum number of bins, such that, the total weight of the items in a bin does not exceed its capacity.
- **Vector Packing Problem (VPP).** Let $I = \{1, \ldots, \mathcal{I}\}$ be a set of items and $D$ a set of dimensions. Assume an unlimited quantity of identical bins with integer positive weight capacities $Q^d$, $d \in D$. Each $i \in I$ has integer non-negative weights $w_i^d \leq Q^d$, $d \in D$. The goal is to pack all items into the minimum possible number of bins, such that, for each dimension, the total weight of the items in a bin does not exceed its capacity.
- **Variable Sized Bin Packing Problem (VSBPP).** Let $I = \{1, \ldots, \mathcal{I}\}$ be a set of $\mathcal{I}$ items and $K = \{1, \ldots, \mathcal{K}\}$ a set of $\mathcal{K}$ bin types. There are $u_k$ bins of type $k \in K$ available, each one having positive integer capacity $Q_k$ and positive integer cost $c_k$. Each $i \in I$ has integer positive weight $w_i \leq Q_{max} = \max_{k \in K} Q_k$. The goal is to pack all items into a least costly set of bins, considering the availability of each bin type and such that the total weight of the items in a bin does not exceed its capacity.

- **Variable Sized Bin Packing Problem with Optional Items (VSBPPOI).** Same as before, except that each item $i \in I$ is associated to a positive integer penalty $p_i$ for not packing it. The goal is find a packing minimizing the costs of the used bins plus the penalties for the non-packed items.

There is a vast literature on those problems, specially for the classic BPP. A comprehensive survey on exact methods for BPP, including an original comparative computational study, was published in 2016 by Delorme et al. [11]. After that, other exact algorithms for BPP were proposed in [10, 42]. Recent exact algorithms for VPP were presented in [6, 19, 43]. The best exact algoriths for VSBPP are those in [23, 1, 17, 4]. The VSBPPOI was less studied, the best exact algorithms for it were presented in [4].

Big advances in the exact solution of Vehicle Routing Problems (VRPs) by Branch-Cut-and-Price (BCP) algorithms have been accomplished in recent years, as surveyed in [9]. A milestone was certainly the Branch-Cut-and-Price (BCP) algorithm of [24], that could solve Capacitated VRP (CVRP) instances with up to 360 customers, a large improvement upon the previous record of 150 customers. That BCP exploits many algorithmic elements introduced by several authors, combining and enhancing them. Improvements of the same magnitude were later obtained for several other variants, like VRP with Time Windows (VRPTW) [25] and Heterogeneous Fleet VRP (HFVRP) [28]. Unhappily, designing and coding each one of those complex and sophisticated BCPs has been a highly demanding task, measured on several work-months of a skilled team. VRPSolver [31] is a software that contains a state-of-the-art BCP algorithm for a very generic model that encompasses most VRP variants found in the literature. Algorithms for particular problems are obtained by defining certain elements in the generic model (using a Julia language interface), in the so-called *specific problem VRPSolver models*. Experiments with VRPSolver models on ten of the most classic VRP variants, including CVRP, VRPTW and HFVRP, show a performance that is competitive or even superior to best specific algorithms for each one of those variants.

This works proposes VRPSolver models for the above mentioned bin packing variants and investigates the performance of the resulting BCP algorithms. As will be shown, even though bin packing problems are not VRPs, VRPSolver can be a quite effective tool for solving them. This is not completely unexpected. Some of the most recent algorithms for bin packing problems, like those in Heßler et al. [19] and Wei et al. [42], are BCP algorithms that clearly borrow ideas from VRP literature. In particular, those algorithms also solve the pricing subproblem as a resource constrained shortest paths problem, using a labeling algorithm, as is usual on VRP.

The theoretical novelty of this paper is related to the branching scheme over accumulated resource consumption by Gélinas et al. [15]. The scheme was originally proposed in the context of an algorithm for a time constrained VRP, but it can be applied in any situation where the pricing subproblem is a resource constrained shortest path and is implemented in VRPSolver. It has the very nice feature of not increasing the pricing complexity in any child node. So, we adopted it in our bin packing VRPSolver models. In principle, we believed that an additional branching scheme, like Ryan and Foster [34] (that makes pricing subproblems harder), would be needed after all accumulated consumption branching alternatives were exhausted. Happily, we could prove that this is not necessary.

This paper is organized as follows. Section 2 reviews the generic VRPSolver model, used to define the specific models given in Section 3. Section 4 contains the proof that the branching on accumulated resource consumption, used in all models, is sufficient. Section 5 presents computational results and comparisons with existing algorithms in the literature. Finally, some additional analysis of the results and future perspectives are provided in Section 6.

# 2 Reviewing the Generic VRPSolver Model

The generic VRPSolver model is a special Mixed Integer Program (MIP) that contains variables associated to resource constrained paths over directed, not necessarily simple, graphs defined

by the user. As the number of such variables is usually huge, they are dynamically priced by solving Resource Constrained Shortest Path (RCSP) problems [20]. Since the integrality of some variables need to be enforced, the MIP is solved by a Branch-and-Price (BP) algorithm. If cuts are also separated the resulting algorithm becomes a Branch-Cut-and-Price (BCP). In particular, if the so-called packing sets are defined, Limited-Memory Rank-1 cuts are automatically separated. This section reviews the VRPSolver model. Some advanced features not used in this paper are omitted, readers interested in knowing them may refer to this detailed reference [30].

## 2.1 Path Generator Graph

All models in this paper use a single path generator graph. So, we simplify the explanation by assuming that the user defines a single graph $G = (V, A)$. She/he should also define: (1) two special vertices in $V$, $v_{\text{source}}$ and $v_{\text{sink}}$; (2) a set $R$ of resources, together with their arc consumptions and accumulated consumption intervals. For each $r \in R$ and $a \in A$, $q_{a,r}$ is the consumption of resource $r$ in arc $a$ and $[l_{a,r}, u_{a,r}]$ is its accumulated resource consumption interval. A path $p = (v_{\text{source}} = v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n = v_{\text{sink}})$ in $G$ is said to be resource constrained if, for every $r \in R$, the accumulated resource consumption $S^p_{j,r}$ at visit $j$, $0 \leq j \leq n$, where $S^p_{0,r} = 0$ and $S^p_{j,r} = \max\{l_{a_j,r}, S^p_{j-1,r} + q_{a_j,r}\}$, does not exceed $u_{a_j,r}$. We remark that the previous definition allows resources to be disposed in order to satisfy the lower bounds $l_{a,r}$ on accumulated consumption of an arc $a$; on the other hand, upper bounds $u_{a,r}$ are strict. An example of a situation where resources can be disposed would be in the VRP with Time Windows problem, where a vehicle can arrive early at a customer and wait (i.e., dispose some time resource) until the opening of its time window. Let $P$ denote the set of all resource constrained paths in $G$. For all $a \in A$ and $p \in P$, let $h^p_a$ indicate how many times arc $a$ appears in path $p$.

## 2.2 Formulation and Mapping

The MIP model is defined by the user as follows. There are variables $x_j$, $1 \leq j \leq n_1$, and variables $y_s$, $1 \leq s \leq n_2$. The first $\bar{n}_1$ $x$ variables and the first $\bar{n}_2$ $y$ variables are defined to be integer. Equations (1a) and (1b) define a general objective function and $m$ general constraints over those variables, respectively. For each variable $x_j$, $1 \leq j \leq n_1$, $M(x_j) \subseteq A$ defines its *mapping* into a non-empty subset of the arcs. Mappings do not need to be disjoint, the same arc can mapped to more than one variable $x_j$. Define $M^{-1}(a)$ as $\{j | 1 \leq j \leq n_1; a \in M(x_j)\}$. As not all arcs need to belong to some mapping, some $M^{-1}$ sets may be empty. For each path $p \in P$, let $\lambda_p$ be a non-negative integer variable. The relation between variables $x$ and $\lambda$ is given by (1c). The values $L$ and $U$ are given lower and upper bounds on number of paths in a solution.

$$\text{Min} \quad \sum_{j=1}^{n_1} c_j x_j + \sum_{s=1}^{n_2} f_s y_s \tag{1a}$$

$$\text{S.t.} \quad \sum_{j=1}^{n_1} \alpha_{ij} x_j + \sum_{s=1}^{n_2} \beta_{is} y_s \geq d_i, \qquad i = 1, \ldots, m, \tag{1b}$$

$$x_j = \sum_{p \in P} \left( \sum_{a \in M(x_j)} h^p_a \right) \lambda_p, \qquad j = 1 \ldots, n_1, \tag{1c}$$

$$L \leq \sum_{p \in P} \lambda_p \leq U, \tag{1d}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad p \in P, \tag{1e}$$

$$x_j \in \mathbb{Z},\ y_s \in \mathbb{Z}, \qquad j = 1, \ldots, \bar{n}_1,\ s = 1, \ldots, \bar{n}_2. \tag{1f}$$

A feasible solution to Formulation (1) is composed of a set of paths, each path $p \in P$ with multiplicity $\lambda_p$ in the solution, and perhaps additional decisions represented by the values assigned to variables $y_s$, $s = 1, \ldots, n_2$. Hence, modelling a problem as (1) requires it to contain structures that can be cast into paths in a properly defined graph. It is preferable that such a

graph has polynomial size. Then, resources should be created to model "intrapath" constraints while global "interpath" constraints, and the objective function, should be modelled as (1b), and (1a), respectively. Note that the values of $x_j$, $j = 1, \ldots, n_1$, are completely defined as a function of the path variables, through the mappings. Thus, these variables are only created for allowing expressing (1a) and (1b).

Eliminating the $x$ variables and relaxing the integrality constraints, the following LP, corresponding to the root node of the BP algorithm, is obtained:

$$\text{Min} \quad \sum_{p \in P} \left( \sum_{j=1}^{n_1} c_j \sum_{a \in M(x_j)} h_a^p \right) \lambda_p + \sum_{s=1}^{n_2} f_s y_s \tag{2a}$$

$$\text{S.t.} \quad \sum_{p \in P} \left( \sum_{j=1}^{n_1} \alpha_{ij} \sum_{a \in M(x_j)} h_a^p \right) \lambda_p + \sum_{s=1}^{n_2} \beta_{is} y_s \geq d_i, \qquad i = 1, \ldots, m, \tag{2b}$$

$$L \leq \sum_{p \in P} \lambda_p \leq U, \tag{2c}$$

$$\lambda_p \geq 0, \qquad p \in P. \tag{2d}$$

Master LP (2) is solved by column generation. Let $\pi_i$, $1 \leq i \leq m$, denote the dual variables of Constraints (2b), $\nu_+$ and $\nu_-$, are the dual variables of Constraints (2c). The reduced cost of an arc $a \in A$ is defined as:

$$\bar{c}_a = \sum_{j \in M^{-1}(a)} c_j - \sum_{i=1}^{m} \sum_{j \in M^{-1}(a)} \alpha_{ij} \pi_i.$$

The reduced cost of a path $p = (v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n) \in P$ is:

$$\bar{c}(p) = \sum_{j=1}^{n} \bar{c}_{a_j} - \nu_+ - \nu_-.$$

So, the pricing subproblems correspond to finding a path $p \in P$ with minimum reduced cost. The above scheme assumes that no additional cuts are being added to the formulation. The interested reader may consult [24] on how cuts can be handled.

## 2.3 Packing Sets and Some Advanced Algorithmic Elements

Let $\mathcal{B} \subset 2^A$ be a collection of mutually disjoint subsets of $A$ such that the constraints:

$$\sum_{a \in B} \sum_{p \in P} h_a^p \lambda_p \leq 1, \quad B \in \mathcal{B}, \tag{3}$$

are satisfied by at least one optimal solution $(x^*, y^*, \lambda^*)$ of Formulation (1). This means that the arcs in each $B \in \mathcal{B}$ can appear at most once in all paths $p \in P$ that are part of some optimal solution. In those conditions, we say that $\mathcal{B}$ defines a collection of *packing sets*. The definition of a proper $\mathcal{B}$ should be done by the user as part of the modeling.

The information given by the packing sets is used by VRPSolver to improve the solution of (1), switching from a basic BP algorithm to an advanced BCP. The following algorithmic elements based on packing sets are used to solve the models described in this article.

### 2.3.1 Limited-Memory Rank-1 Cuts

The Rank-1 Cuts (R1Cs) [27] are a generalization of the Subset Row Cuts proposed by Jepsen et al. [21]. In the VRPSolver context they are further generalized as follows. Consider a collection of packing sets $\mathcal{B}$ and non-negative multipliers $\rho_B$ for each $B \in \mathcal{B}$. A Chvátal-Gomory rounding of constraints (3) yields:

$$\sum_{p \in P} \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \sum_{a \in B} h_a^p \right\rfloor \lambda_p \leq \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \right\rfloor. \tag{4}$$

R1Cs are potentially strong, but each added cut makes the pricing subproblems significantly harder. The limited memory technique [26] is essential for mitigating that negative impact.

### 2.3.2 Path Enumeration

The path enumeration technique was proposed by Baldacci et. al. [3], and later improved by Contardo and Martinelli [8]. It consists in trying to enumerate into a table all paths in $P$ that can possibly be part of an improving solution. After a successful enumeration, the corresponding pricing subproblem can be solved by inspection, saving time. If the enumeration has already succeeded and the total number of paths in the tables is not too large (say, less than 10,000) the overall problem may be finished by a standard MIP solver, which often saves a lot of time.

From time to time, VRPSolver tries to enumerate all paths $p$ without more than one arc in the same packing set, and with reduced cost $\bar{c}(p)$ smaller than the current gap $UB - LB$, where $UB$ is the best known integer solution cost, and $LB$ the value of the current linear relaxation. Moreover, if two paths $p$ and $p'$ lead to variables $\lambda_p$ and $\lambda'_p$ with identical coefficients in (2b)–(2c), the one with larger cost is dropped.

### 2.3.3 Branching

Branching over $x$ and $y$ variables (or over linear expressions defined over them) is simple and does not change the structure of the pricing subproblem. In many models this is sufficient for correctness. For example, in a Capacitated VRP model where $x_{ij}$ variables indicate whether a vehicle travels from point $i$ to point $j$ (like in [31]), if all $x$ variables are integer then they correspond to a correct solution, there is no need to even check the integrality of the $\lambda$ variables. However, there are models, including all in this paper, where this does not happens and constraints (1e) need to be explicitly enforced. Branching over individual $\lambda$ variables should be avoided due to a big negative impact in the pricing and also due to highly unbalanced branch trees [40].

VRPSolver has the option of branching using a generalization of the Ryan and Foster rule [34]. Choose two distinct sets $B$ and $B'$ in $\mathcal{B}$. Let $P(B, B') \subseteq P$ be the subset of the paths that contain arcs in both $B$ and $B'$. The branch is over the value of $\sum_{p \in P(B,B')} \lambda_p$. The branch trees are much more balanced. However, Ryan and Foster branching scheme changes the structure of the pricing subproblem, sometimes increasing a lot its difficulty.

VRPSolver also implements a branching scheme similar to the one proposed by Gélinas et al. [15] for time constrained routing problems. It can be described as follows. Assume that all consumptions and accumulated consumption intervals are integer. For a chosen $B \in \mathcal{B}$, $r \in R$ and for a certain threshold value $t^*$: in the left child make $u_{a,r} = t^* - 1$, for all $a \in B$; in the right child make $l_{a,r} = t^*$, for all $a \in B$. In other words, the branch is over the accumulated consumption of resource $r$ on arcs in $B$. This branching has the nice feature of not increasing the pricing difficulty. However, it is not sufficient for general Formulation (1), since some fractional $\lambda$ solutions can not be eliminated by it. The main theoretical contribution of this paper (in Section 4) is a proof that the branching scheme over accumulated resource consumption is sufficient for the proposed bin packing models.

# 3 VRPSolver Models for Bin Packing problems

Now we present the specific VRPSolver models corresponding to each of the problems addressed in this paper.

### 3.1 Vector Packing (VPP) / Bin Packing (BPP)

The following model is valid for the VPP, the classic BPP corresponds to the case where $|D| = 1$:

**VRPSolver Model for VPP:** Graph $G = (V, A)$, where $V = \{v_i : i \in I \cup \{0\}\}$ and $A = \{a_{i+} = (v_{i-1}, v_i), a_{i-} = (v_{i-1}, v_i) : i \in I\}$; $v_{\text{source}} = v_0$, $v_{\text{sink}} = v_{\mathcal{I}}$; $R = D$; $q_{a_{i+},d} = w_i^d, q_{a_{i-},d} = 0, i \in I, d \in D$; $[l_{a,d}, u_{a,d}] = [0, Q^d]$, $a \in A, d \in D$. Continuous variables $x_i$, $i \in I \cup \{0\}$. The formulation is:

$$\text{Min} \quad x_0 \tag{5a}$$
$$\text{S.t.} \quad x_i = 1, \qquad i \in I; \tag{5b}$$

$M(x_0) = \{a_{1+}, a_{1-}\}$, $M(x_i) = \{a_{i+}\}, i \in I$; $L = 0$, $U = \infty$. $\mathcal{B} = \cup_{i \in I}\{\{a_{i+}\}\}$. Branching over accumulated resource consumption.

The path generator graph is depicted in Figure 1. For each item $i \in I$, there is an arc $a_{i+}$ with consumptions $w_i^d$, $d \in D$, and another arc $a_{i-}$ with zero consumptions. It can be seen that there is a one-to-one correspondence between resource constrained paths in $P$ and solutions of the $|D|$-dimensional binary knapsack problem defined by $\{z \in \{0,1\}^{\mathcal{I}} : \sum_{i \in I} w_i^d z_i^d \leq Q^d, d \in D\}$, that also correspond to the possible ways of packing items into a bin. Variables $x_i$, $i \in I$, indicate if item $i$ is packed. As all items must be packed, they are fixed to 1 in (5b). Each variable $x_i$ is mapped to arc $a_{i+}$. So, Constraints (5b) are equivalent to saying that the solution should contain exactly one path in $P$ passing by each arc $a_{i+}$. Variable $x_0$ is mapped to both $a_{1+}$ and $a_{1-}$. As every path in $P$ passes by exactly one of those arcs, $x_0$ is put in the objective function (5a) for counting the number of used paths (bins).

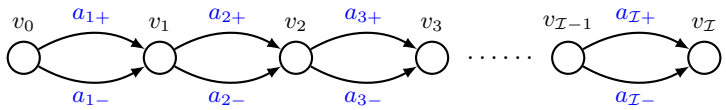


Figure 1: Path generator graph for VPP/BPP.

Consider the BPP instance with $\mathcal{I} = 4$, $w_1 = 7$, $w_2 = 8$, $w_3 = 9$, $w_4 = 11$, and $Q = 17$. The set $P$ has 8 paths: $p_1 = (a_{1-}, a_{2-}, a_{3-}, a_{4-})$, $p_2 = (a_{1+}, a_{2-}, a_{3-}, a_{4-})$, $p_3 = (a_{1-}, a_{2+}, a_{3-}, a_{4-})$, $p_4 = (a_{1-}, a_{2-}, a_{3+}, a_{4-})$, $p_5 = (a_{1-}, a_{2-}, a_{3-}, a_{4+})$, $p_6 = (a_{1+}, a_{2+}, a_{3-}, a_{4-})$, $p_7 = (a_{1+}, a_{2-}, a_{3+}, a_{4-})$, $p_8 = (a_{1-}, a_{2+}, a_{3+}, a_{4-})$. The complete formulation (corresponding to Formulation (1)) for that instance is:

$$\text{Min} \quad x_0 \tag{6a}$$
$$\text{S.t.} \quad x_1 = 1, \tag{6b}$$
$$x_2 = 1, \tag{6c}$$
$$x_3 = 1, \tag{6d}$$
$$x_4 = 1, \tag{6e}$$
$$x_0 = \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8, \tag{6f}$$
$$x_1 = \lambda_2 + \lambda_6 + \lambda_7, \tag{6g}$$
$$x_2 = \lambda_3 + \lambda_6 + \lambda_8, \tag{6h}$$
$$x_3 = \lambda_4 + \lambda_7 + \lambda_8, \tag{6i}$$
$$x_4 = \lambda_5, \tag{6j}$$
$$0 \leq \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8, \tag{6k}$$
$$\lambda \in \mathbb{Z}_+^8 \tag{6l}$$

A possible optimal solution to (6) would have $\lambda_4 = \lambda_5 = \lambda_6 = 1$ (the corresponding paths are depicted in Figure 2) and the remaining $\lambda$ variables equal to zero.

Eliminating the $x$ variables and relaxing the integrality, the master LP (corresponding to (2)) that should be solved by column generation is obtained:

$$\begin{aligned}
\text{Min} \quad & \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 && \text{(7a)}\\
\text{S.t.} \quad & \lambda_2 + \lambda_6 + \lambda_7 = 1, && \text{(7b)}\\
& \lambda_3 + \lambda_6 + \lambda_8 = 1, && \text{(7c)}\\
& \lambda_4 + \lambda_7 + \lambda_8 = 1, && \text{(7d)}\\
& \lambda_5 = 1, && \text{(7e)}\\
& \lambda \geq 0 && \text{(7f)}
\end{aligned}$$

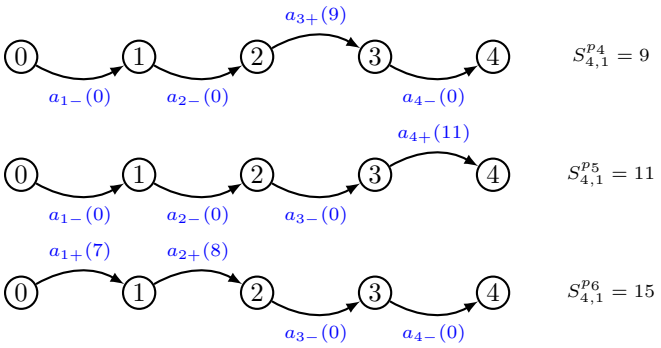


Figure 2: Paths used on an integer solution of (6). The arc consumptions and the accumulated resource consumption at the end of each path are also indicated.

## 3.2 Variable Sized Bin Packing with Optional Items (VSBPPOI)

We present the model for the more general VSBPPOI, the model for the VSBPP without optional items is easily obtained by setting sufficiently large penalties or by removing some variables from the model.

**VRPSolver Model for VSBPPOI:** Graph $G = (V, A)$, where $V = \{v_i : i = 0, \ldots, \mathcal{I}+1\}$ and $A = (A_1 = \{a_{i+} = (v_{i-1}, v_i), a_{i-} = (v_{i-1}, v_i) : i = 1, \ldots, \mathcal{I}\}) \cup (A_2 = \{a_k = (v_{\mathcal{I}}, v_{\mathcal{I}+1}) : k \in K\})$; $v_{\text{source}} = v_0$, $v_{\text{sink}} = v_{\mathcal{I}+1}$; $R = \{1\}$; $q_{a_{i+},1} = w_i, q_{a_{i-},1} = 0, i \in I, q_{k,1} = 0, k \in K$; $[l_{a,1}, u_{a,1}] = [0, Q_{\max}]$, $a \in A_1$, where $Q_{\max} = \max_{k \in K} Q_k$; $[l_{a_k,1}, u_{a_k,1}] = [0, Q_k]$, $k \in K$. Integer variables $x_i$ and $s_i$, $i \in I$; integer variables $z_k$, $k \in K$. The formulation is:

$$\begin{aligned}
\text{Min} \quad & \sum_{k \in K} c_k z_k + \sum_{i \in I} p_i s_i && && \text{(8a)}\\
\text{S.t.} \quad & x_i + s_i = 1, && i \in I, && \text{(8b)}\\
& z_k \leq u_k, && k \in K, && \text{(8c)}\\
& s_i \geq 0, && i \in I; && \text{(8d)}
\end{aligned}$$

$M(x_i) = \{a_{i+}\}, i \in I$; $M(z_k) = \{a_k\}, k \in K$; $L = 0$, $U = \infty$. $\mathcal{B} = \cup_{i \in I}\{\{a_{i+}\}\}$. We first branch on expressions $\sum_{i \in I} s_i$ and $\sum_{k \in K} \frac{c_k}{g} z_k$, where $g$ is the greatest common divisor of $\{c_k\}_{k \in K}$. If both these expressions are integer, we branch over $s$ and $z$ variables as well as over accumulated resource consumption.

The path generator graph is depicted in Figure 3. The graph differs from that in the previous BPP/VPP model by having one more vertex and extra arcs $a_k$, $k \in K$. The paths in $P$ passing

by an arc $a_k$ are associated to the possible packings in bin type $k$. Each variable $x_i$, $i \in I$, mapped to arc $a_{i+}$, indicates if item $i$ is packed. Unlike in the previous model, those variables are not fixed to 1. Each variable $z_k$, $k \in K$, mapped to arc $a_k$, counts how many bins of type $k$ are used in the solution. Variables $s$ are not mapped.

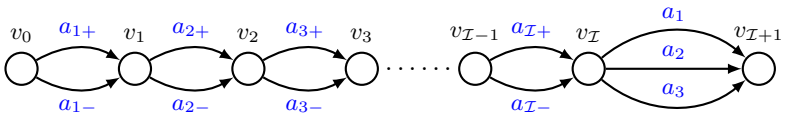


Figure 3: Path generator graph for VSBPPOI with three bin types.

The expression $\sum_{i \in I} s_i$ corresponds to the number of items that are not packed. Branching on the value of that expression can only be applied on VSBPPOI instances. The expression $\sum_{k \in K} \frac{c_k}{g} z_k$ is proportional to the total cost of the bins used in the solution. On VSBPP instances (without optional items), branching down on the value of this expression yields an infeasible left child node; in the right child the node lower bound is certainly increased at least to the next multiple of $g$. The rounding up of lower bounds to the next multiple of the greatest common divisor of the bin costs is standard in published VSBPP algorithms. Even on VSBPPOI instances, where the left child is feasible, this branching is usually quite good in increasing lower bounds.

# 4 Branching over Accumulated Resource Consumption

The classic lower bound for the BPP by Gilmore and Gomory [16] is obtained by column generation, solving binary knapsack problems in the pricing. The problem is (weakly) NP-hard, but there are some advanced knapsack algorithms that perform very well in practice [32]. However, implementing a BP algorithm over Gilmore and Gomory can be tricky.

Branching directly over the variables of the model (as done in [5]) leads to very unbalanced search trees, fixing a variable corresponding to a certain packing of items to a bin to one is strong, but fixing it to zero is much less likely to move the lower bounds. Moreover, fixing variables to zero change the structure of the pricing, each fixing making it harder.

Another alternative, first used in [39], is to apply Ryan and Foster scheme [34], choosing a pair of items $i$ and $j$. In the left child, $i$ and $j$ should be packed in the same bin. This does not change the pricing structure, the items are simply merged into a single item. In the right child, $i$ and $j$ can not be packed in the same bin. This changes the structure of the pricing, that becomes a binary knapsack with conflicts, which is a strongly NP-hard problem.

The BPP model presented in Section 3.1 is equivalent to Gilmore and Gomory model, however VRPSolver uses the Resource Constrained Shortest Path problem as pricing subproblem. RCSPs are solved using a labeling algorithm (see [35] for details), which is a dynamic programming where reachable states are represented as labels. The practical efficiency of a labeling algorithm depends on the concept of *dominance*. Let $L_1$ and $L_2$ be the labels corresponding to partial paths $p_1$ and $p_2$ in $G$, starting in $v_{source}$ and ending at the same vertex $v$ of $V$. If the accumulated resource consumption of $p_1$ is not larger than the accumulated resource consumption of $p_2$ (for all $r \in R$) and the reduced cost of $p_1$ is smaller than the reduced cost of $p_2$, then label $L_2$ is dominated by $L_1$ and can be removed. This dominance rule is correct because every extension of $p_2$ into a complete path in $P$, if applied to $p_1$, would produce a complete path in $P$ with smaller reduced cost. We remark that the "smaller-consumption-is-better" rule works because the RCSP definition permits resources to be disposed, if this is needed to satisfy lower bounds on accumulated consumption.

A potential advantage of solving the pricing subproblem as a RCSP is that the branching over accumulated resource consumption, which never changes the pricing structure, can be used. That branching scheme was proposed by Gélinas et al. [15] for time constrained routing problems. However, the authors did not prove its sufficiency. In fact, they proposed using a second

branching scheme for the cases when the current fractional solution could not be eliminated by branching over accumulated resource consumption in both children nodes. The main theoretical result of this work is Theorem 1. For the sake of simplicity, it is stated only for BPP. However, we will show later that the sufficiency result also holds for all models in Section 3.

**Definition 1.** *A branching is* effective *if it cuts the current fractional solution in both child nodes.*

**Definition 2.** *A branching scheme is* sufficient *if, given a fractional solution, it provides either an effective branching or a polynomial algorithm to convert that fractional solution into an integer solution with the same cost.*

Consider as example the instance described in Section 3.1 and its linear relaxation (7), having solution $\lambda_5 = 1$, $\lambda_6 = \lambda_7 = \lambda_8 = 0.5$, the remaining variables with value zero. The accumulated resource consumptions for the paths associated to the fractional variables are: $S^{p_6}_{1,1} = 7$, $S^{p_6}_{2,1} = S^{p_6}_{3,1} = S^{p_6}_{4,1} = 15$, $S^{p_7}_{1,1} = S^{p_7}_{2,1} = 7$, $S^{p_7}_{3,1} = S^{p_7}_{4,1} = 16$, $S^{p_8}_{1,1} = 0$, $S^{p_8}_{2,1} = 8$, $S^{p_8}_{3,1} = S^{p_8}_{4,1} = 17$. It is not possible to branch effectively over item 3, corresponding to arc $a_{3+}$. This arc appears in paths $p_7$ and $p_8$. Choosing $t^* = 17$, we would have intervals $[0, 16]$ and $[17, 17]$ for the accumulated resource consumption of arc $a_{3+}$ in the left and right child nodes, respectively. The first interval would indeed eliminate $p_8$, but the second would eliminate neither $p_7$ (because 1 unit of resource can be dropped to make $S^{p_7}_{3,1} = 17$) nor $p_8$. On the other hand, it is possible to branch effectively over item 2. Choosing $t^* = 10$ would give intervals $[0, 9]$ and $[10, 17]$ for $a_{2+}$. The first interval would eliminate $p_6$, while the second would eliminate $p_8$. So, the fractional solution would be cut in both branches.

**Lemma 1.** *Suppose that one sets tight intervals on accumulated resource consumption for all items, that is, $l_{a_{i+},1} = u_{a_{i+},1}$, $i \in I$. The linear relaxation of the BPP model (corresponding to (2)) restricted to the paths in $P$ that respect those tight intervals has an optimal integer solution.*

*Proof.* Define the following Minimum Cost Flow Circulation ($MCFC$) problem over graph $H = (\{0, \ldots, 2\mathcal{I}+1\}, F)$, where $F = \{(0, 2i-1), (2i-1, 2i), (2i, 2\mathcal{I}+1 : i \in I\} \cup \{(2i_1, 2i_2 - 1) : i_1, i_2 \in I,\ i_1 < i_2,\ l_{a_{i_1+}} + w_{i_2} \leq l_{a_{i_2+}}\} \cup \{(2\mathcal{I}+1, 0)\}$. All arcs in $F$ cost zero, except by $(2\mathcal{I}+1, 0)$ that costs 1. The flow $f_a$ in each arc $a \in F$ can assume any non-negative value, except by arcs in $\{(2i-1, 2i) : i \in I\}$ where the flow is fixed to 1. Let $z(MCFC)$ be the optimal solution value of $MCFC$. Let $BPP_{LP}(P')$ be the linear relaxation of the BPP model restricted to $P'$, the subset of $P$ formed by the paths that respect the tight intervals. Let $z(BPP_{LP}(P'))$ be its optimal solution value.

There is a one-to-one correspondence between paths in $P'$ and cycles in $F$. Let $\bar{\lambda}$ be an optimal solution (integer or fractional) of $BPP_{LP}(P')$. Start with zero flow for all arcs in $F$. For each $p \in P'$, add the value $\bar{\lambda}_p$ to the flow of all arcs of the cycle in $F$ corresponding to $p$. The resulting flow is a solution of $MCFC$ with value $z(BPP_{LP}(P'))$. Conversely, a cycle decomposition of an optimal solution of $MCFC$ yields a solution of $BPP_{LP}(P')$ with value $z(MCFC)$. So, $z(BPP_{LP}(P')) = z(MCFC)$. Moreover, the Flow Integrality Theorem asserts that $MCFC$ has an optimal solution where all flows are integers. That integer solution will yield an optimal integer solution for $BPP_{LP}(P')$. □ □

In order to exemplify Lemma 1, consider the same BPP instance from Section 3.1 and suppose that intervals are set to [8,8], [8,8], [17,17] and [14,14], for items 1, 2, 3 and 4, respectively. Graph $H$ is depicted in Figure 4, $P' = P \setminus \{p_6\}$, and $z(BPP_{LP}(P')) = z(MCFC) = 3$. A minimum cost integer flow circulation is has arcs $f_{01}$, $f_{12}$, $f_{25}$, $f_{56}$, $f_{69}$, $f_{03}$, $f_{34}$, $f_{49}$, $f_{07}$, $f_{78}$, and $f_{89}$ with value 1 and $f_{90} = 3$. This flow corresponds to an integer solution having $\lambda_3 = \lambda_5 = \lambda_7 = 1$.

**Theorem 1.** *The branching scheme over accumulated resource consumption is sufficient for the VRPSolver Model for BPP.*

*Proof.* In this context, a branching over an item $i \in I$ and threshold value $t^*$ is effective if there exists a pair of paths $p_1$ and $p_2$ passing by $a_{i+}$, where $\lambda_{p_1}$ and $\lambda_{p_2}$ have positive value in the

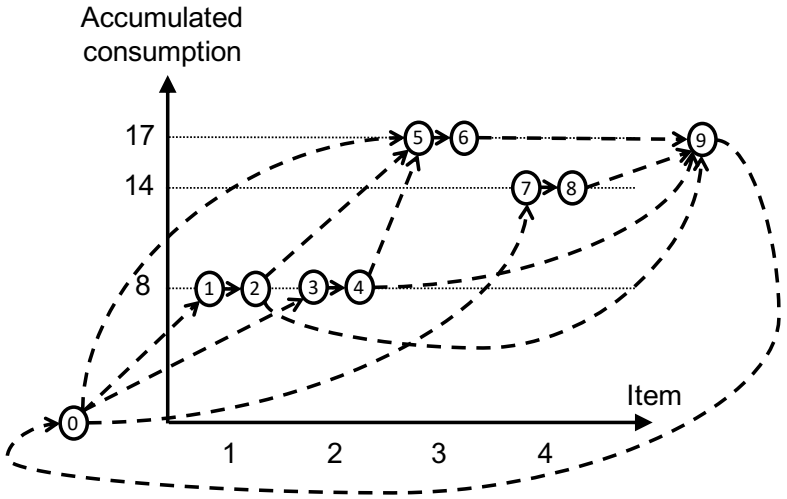


Figure 4: Example of a network obtained from a BPP instance with tight consumption bounds

current fractional relaxation, and such that: if $u_{a_{i+},1}$ is set to $t^* - 1$ (left child) then $p_1$ remains feasible but $p_2$ not; if $l_{a_{i+},1}$ is set to $t^*$ (right child) then $p_2$ remains feasible but $p_1$ not. Assume that there is no effective branching. This means for each item $i \in I$ and for every threshold $t^*$, all paths (including, of course, the paths that do not pass by $a_{i+}$) with positive fractional variable would remain feasible either in the left or in the right child. Consider some $i \in I$, and let $t_i^{**}$ be the maximum threshold such that all those paths would remain feasible in the right child, threshold $t_i^{**} + 1$ would make all those paths to be feasible in the left child. This means that all paths with positive fractional value are feasible for *both* intervals $[0, t_i^{**}]$ and $[t_i^{**}, Q]$ on the accumulated consumption of $a_{i+}$, so they are all feasible for tight interval $[t_i^{**}, t_i^{**}]$. Repeating the reasoning for each $i \in I$, one at a time, we can conclude that all paths with positive fractional value would remain feasible even if all intervals are tightened to $[t_i^{**}, t_i^{**}]$. Then, by Lemma 1, the current fractional solution has the same cost of an optimal integer solution. In fact, in order to obtain that solution itself it may be necessary to solve (in polynomial time) the $MCFC$ instance defined in the proof of Lemma 1. □ □

We now sketch how to adapt the above proof for obtaining similar sufficiency results for the other variants considered.

- The branching scheme over accumulated resource consumption is also sufficient for the more generic VRPSolver Model for VPP. The reasoning in Lemma 1 and Theorem 1 remains essentially the same, showing that a fractional solution without an effective branching would also be a solution to a restricted problem where the intervals on accumulated consumption associated to each item $i \in I$ are tight for each resource $d \in D$.

- The same branching scheme also suffices for the VRPSolver Model for VSBPP. The proof only differs in Lemma 1, where graph $H$ would have vertex-set $\{0, \ldots, 2\mathcal{I} + \mathcal{K}\}$ and arc-set $F = \{(0, 2i-1), (2i-1, 2i) : i \in I\} \cup \{(2i, 2\mathcal{I}+k) : k \in K, i \in I, l_{a_{i+},1} \leq Q_k\} \cup \{(2i_1, 2i_2) : i_1, i_2 \in I,\ i_1 < i_2,\ l_{a_{i_1+},1} + w_{i_2} \leq l_{a_{i_2+},1}\} \cup \{(2\mathcal{I}+k, 0) : k \in K\}$. Arcs $(2\mathcal{I}+k, 0)$, $k \in K$ would have flow capacity $u_k$ and cost $c_k$.

- Finally, the branching scheme for the VRPSolver Model for VSBBPOI of branching over $z$ variables and over accumulated resource consumption is sufficient. This is true because if all $z$ variables are integer, then the same proof used for the VSBPP guarantees the existence of an effective branching over accumulated resource, unless the current fractional solution can be converted into an integer solution with the same cost.

# 5 Computational Experiments

The models presented in Section 3 were implemented on VRPSolver [31, 30], available for free academic use at `https://vrpsolver.math.u-bordeaux.fr`. VRPSolver solves RCSP pricing subproblems with the bucket graph based labeling algorithm proposed in [35] and applies automatic dual price smoothing stabilization [29]. CPLEX 12.8 is used for solving linear programs and MIPs. The experiments were run on a 2 Deca-core Ivy-Bridge Haswell Intel Xeon E5-2680 v3 server running at 2.50 GHz. The 128 GB of available RAM was shared between 8 copies of the algorithm running in parallel on the server. Each instance is solved by one copy of the algorithm using a single thread.

## 5.1 Bin Packing Problem

For the experiments on the BPP we use the hardest (according to [11]) classes of literature instances: "Falkenauer T" [13], "Wäscher" [41], "Hard28" [37], as well as "AI" and "ANI" instances recently proposed in [11].

The default parameterization of VRPSolver (see [30]) is changed in the following way:

- The number of buckets per vertex is set to 200, and it is not dynamically adjusted.
- The bidirectional variant of the labeling algorithm is applied when solving the pricing both heuristically and exactly.
- At most 100 columns are generated at every iteration of column generation.
- When applying the path enumeration technique, the number of enumerated paths is limited to $2 \cdot 10^6$, i.e. path enumeration is interrupted if this number is exceeded.
- The node is finished by the MIP solver if the number of enumeration paths is less or equal to $10^5$.
- 3- and 4-row limited-memory rank-1 cuts are separated, i.e. they are obtained by Chvátal-Gomory rounding of 3 and 4 constraints (3), respectively.
- The cut generation tailing-off threshold is set to 5, i.e. the cut generation is stopped when the primal-dual gap is decreased by less than 2% after 5 cut separation rounds.
- The safe lower bound technique similar to the one proposed in [18] is applied to assure the validity of the column generation bound.
- At most 20 branching candidates are considered during the strong branching.

For each instance, we use initial primal bound which is equal to the rounded up value of the column generation lower bound plus 1 unit. There is a long-standing conjecture that the optimal solution value of a BPP instance is never larger than this. Solutions with these objective values are easily obtainable by simple heuristics. The pure diving heuristic [36] is used to improve initial primal bound. It is executed at every node of the search tree unless its depth is greater that 10.

We compare the results of VRPSolver with the best approaches in the literature. These are the following:

**BelSch06** branch-cut-and-price algorithm proposed in [5] and executed at an Intel Xeon 3.10 GHz processor in [11]

**DelIori20** enhanced pseudo-polynomial formulation proposed in [10], also executed at an Intel Xeon 3.10 GHz processor,

**WLBL20** branch-cut-and-price algorithm proposed in [42] and executed at an Intel Xeon E5-1603 2.80 GHz.

The results are shown in Table 1. In the first column, the name of the data set is given. Second column gives the number of items in the instances of this data set. Then for each algorithm, we show the number of instances solved within the time limit and the average solution time. The time limit is 10 minutes for the first three classes of instances, and 1 hour for instances in classes AI and ANI. For unsolved instances, the solution time is set to the time limit. We mark in bold the best results for each data set, considering first the number of solved instances and then the average time in case of ties. It can be seen that VRPSolver is able to solve

| | | **VRPSolver** | | | **BelSch06** | | **DelIori20** | | **WLBL20** | |
|---|---|---|---|---|---|---|---|---|---|---|
| Data set | # Items | Opt. | Time | Nodes | Opt. | Time | Opt. | Time | Opt. | Time |
| Falken.T | 60–501 | 80/80 | 22 | 1.0 | 80/80 | 56 | 80/80 | **1** | 80/80 | 2 |
| Wäscher | 57–239 | 17/17 | 91 | 1.0 | 17/17 | **1** | 17/17 | 41 | 17/17 | 16 |
| Hard28 | 160–200 | 28/28 | 14 | 1.3 | 28/28 | 8 | 28/28 | **4** | 27/28 | 9 |
| AI | 201 | 50/50 | 72 | 3.8 | 50/50 | 144 | 50/50 | 9 | 50/50 | **4** |
| | 402 | **47/50** | 403 | 14.3 | 45/50 | 699 | 40/50 | 1205 | 45/50 | 398 |
| | 600 | **35/50** | 1458 | 4.8 | 21/50 | 2539 | – | – | 27/50 | 1760 |
| | 801 | **22/50** | 2918 | 2.3 | 0/50 | 3600 | – | – | 15/50 | 2766 |
| | 1002 | 0/50 | 3600 | 1.0 | – | – | – | – | **2/50** | 3546 |
| ANI | 201 | 50/50 | 17 | 1.3 | 50/50 | 144 | 50/50 | 50 | 50/50 | **14** |
| | 402 | **50/50** | 96 | 1.3 | 1/50 | 3556 | 47/50 | 2704 | 45/50 | 436 |
| | 600 | **1/50** | 3565 | 10.8 | 0/50 | 3600 | – | – | 0/50 | 3600 |
| | 801 | 0/50 | 3600 | 1.8 | 0/50 | 3600 | – | – | 0/50 | 3600 |
| | 1002 | 0/50 | 3600 | 1.0 | – | – | – | – | 0/50 | 3600 |

Table 1: Comparison of VRPSolver with best approaches from the literature on BPP instances

more instances to optimality than any other algorithm. However, the average solution time is significantly worse than competitors for the first three classes of instances. For the two most difficult classes AI and ANI, our approach clearly showed the best results. Algorithm WLBL20 is close to VRPSolver both in terms of the number of solved instances and the solution time in seconds. This is not surprising as WLBL20 is a branch-cut-and-price algorithm similar to the one used in VRPSolver, but with a specialized implementation. The main differences are that VRPSolver i) uses bi-directional labeling algorithm for the pricing problem, ii) employs stabilization and enumeration, iii) uses 3- and 4-row limited-memory rank-1 cuts, whereas WLB uses only 3-row full-memory cuts, and iv) uses branching over accumulated resource consumption instead of Ryan&Foster branching.

The bottleneck of our algorithm for solving instances of classes AI and ANI with 600 items or more is the LP solver numerical tolerance. For such instances, sometimes a column added to the master problem does not enter in the basis, in spite of having a negative reduced cost according to the current optimal dual solution provided by CPLEX. This may happen even if the reduced cost tolerance in CPLEX is set to its minimum value $10^{-9}$. In those cases, the column generation procedure is stopped and the safe Lagrangean bound is used. The safe bound is a only a little weaker than the potential bound that would be obtained by solving the master LP to the end. Yet, this may make a lot of difference. It is quite frequent on those hard BPP instances that the lower bound of a node is very close to be one unit away from the primal upper bound. The use of a slightly worse lower bound may be enough to prevent the pruning of that node.

In Table 2, we compare branching over accumulated resource consumption and Ryan and Foster branching. The comparison is done on classes of instances for which average number of nodes is greater than one, i.e. branching is needed for at least one instance in the class. It can be seen that one can solve more instances when employing the branching over accumulated resource consumption, and the average solution time is also decreased. The main reason is that Ryan and Foster branching is "non-robust". Additional binary resources should be added to the resource constrained shortest path pricing problem to take into account Ryan and Foster

| | | Res. cons. branching | | | Ryan&Foster branching | | |
|---|---|---|---|---|---|---|---|
| Data set | # Items | Opt. | Time | Nodes | Opt. | Time | Nodes |
| Hard28 | 160–200 | 28/28 | **14** | 1.3 | 28/28 | 15 | 1.4 |
| AI | 201 | 50/50 | 72 | 3.8 | 50/50 | **70** | 2.6 |
| | 402 | **47/50** | 403 | 14.3 | 43/50 | 668 | 3.0 |
| | 600 | **35/50** | 1458 | 4.8 | 34/50 | 1530 | 2.9 |
| | 801 | **22/50** | 2918 | 2.3 | 21/50 | 2932 | 1.5 |
| ANI | 201 | 50/50 | 17 | 1.3 | 50/50 | 17 | 1.2 |
| | 402 | 50/50 | **96** | 1.3 | 50/50 | 116 | 1.7 |
| | 600 | 1/50 | **3565** | 10.8 | 1/50 | 3568 | 7.9 |
| | 801 | 0/50 | 3600 | 1.8 | 0/50 | 3600 | 1.6 |

Table 2: Comparison of branching strategies

branching constraints. Thus the pricing problem takes significantly more time to be solved, and less nodes can be explored within the time limit.

| | | RCSP pricing | | | Knapsack pricing | | |
|---|---|---|---|---|---|---|---|
| Data set | # Items | Time | Iters. | Cols. | Time | Iters. | Cols. |
| Falken.T | 60–501 | 6 | 84 | 7180 | 2 | 597 | 688 |
| Wäscher | 57–239 | 12 | 87 | 8045 | 3 | 495 | 519 |
| Hard28 | 160–200 | 4 | 91 | 7443 | 2 | 654 | 749 |
| AI | 201 | 12 | 159 | 14228 | 4 | 979 | 1061 |
| | 402 | 75 | 319 | 29137 | 33 | 2227 | 2397 |
| | 600 | 277 | 561 | 52353 | 117 | 3313 | 3564 |
| | 801 | 1132 | 993 | 94073 | 322 | 4051 | 4835 |
| ANI | 201 | 13 | 161 | 14023 | 4 | 1042 | 1125 |
| | 402 | 76 | 318 | 28582 | 33 | 2282 | 2453 |
| | 600 | 286 | 554 | 51049 | 117 | 3355 | 3607 |
| | 801 | 1146 | 993 | 93886 | 322 | 4550 | 4884 |

Table 3: Comparison of pricing algorithms for solving the first linear relaxation

The first linear relaxation of the BPP model (corresponding to (2)), before cuts are added or branching is performed, is equivalent to Gilmore and Gomory relaxation and can be solved using a knapsack algorithm in the pricing. Table 3 presents a comparison of solving that first relaxation by using the VRPSolver RCSP labeling algorithm in the pricing with the use of a high-performance specialized algorithm for the binary knapsack problem. For the latter we have chosen the algorithm by Pisinger [33]. We skip instances with 1000 items as for some of these instances, the column generation algorithm did not converge in 1 hour when the pricing is solved by the labeling algorithm. The first two columns are as in the previous tables. Then in next columns, for each algorithm, we show the average solution time, average number column generation iterations, and the average number of generated columns. As can be seen, using a specialized algorithm for the knapsack problem makes that column generation procedure two to four times faster. The number of iterations is larger as at most one column per iteration is generated. When using the labeling algorithm, at most 100 columns are generated on every iteration. However, each iteration, including the solution of a larger LP, is much more expensive.

In spite of being less efficient for the first column generation, the use of the labeling algorithm for the RCSP in the overall branch-cut-and-price algorithm still has many advantages, allowing the use of important algorithmic elements: rank-1 cuts, path enumeration and branching over accumulated resource consumption. However, a hybrid pricing strategy that uses Pisinger's code for the first column generation, while the subproblem structure is still a knapsack, would indeed save some time. In some cases the saving would be significant. For example, for the AI instances with 201 items the average total time would decrease from 17 to 6 seconds. We preferred not do that in our main experiments because the publicly available version of the VRPSolver does

not have the feature of using external algorithms for the pricing, its users would not be able to reproduce the reported results.

Another note concerns the set-partitioning constraints (5b). It could be advantageous to define set-covering constraints of format $x_i \geq 1$ instead, so the dual variables would never be negative. However, our preliminary experiments showed that the solution time of the first column generation would be reduced by only 10-20%, due to the fact that VRPSolver already uses a strong stabilization mechanism. No gains were observed in the remaining of the algorithm, after cuts, enumeration, diving heuristics or branching starts to be performed. Thus, the overall improvements were not significant. We prefer to keep the set-partitioning constraints because they are more intuitive to the average VRPSolver. For similar reasons, the dual cuts from [38] do not improve results significantly and were not used in the reported experiments.

A last note concerns the item ordering in the path generator graph (see Figure 1). In our tests, the items followed the order in which they appear in the original instance files, sorted in non-increasing order of weights. We have experimented with two different orders: a random order and the alternating order, in which the first item (the one with largest weight) goes first in the graph, the second item (the second largest weight goes last in the graph), the third item goes second in the graph and so on. These experiments show that the item ordering does not have any significant impact on the results. The reason is that the number of non-dominated labels in the exact labeling algorithm during the first column generation convergence is close to $\mathcal{I}.\mathcal{Q}$, the number of states in the standard dynamic programming algorithm for the binary knapsack problem. Therefore, there is no sparsity to be explored by changing the order of items.

## 5.2 Vector Packing Problem

For the experiments on the VPP we use classic 2-dimensional instances generated in [7]. These instances have from 25 to 200 items. We have also used the 20-dimensional instances obtained in [6] by aggregating ten 2-dimensional ones. The parameterisation of VRPSolver is similar to the one used for the BPP instances except by the following changes.

- The number of buckets per vertex is set to 2000 in the labeling algorithm.
- A labeling heuristic is used for the heuristic pricing in which each bucket contains at most 8 non-dominated labels.
- Rank-1 cuts are not generated.
- When applying the path enumeration technique, the number of labels is limited to $10^5$. If this number is exceeded, the path enumeration is interrupted.

We do not use initial upper bounds. In order to obtain primal solutions, the diving heuristic embedded in VRPSolver is used only at the root node, with Limited Discrepancy Search [36] having parameterisation $\chi^{\text{depth}} = 2$, $\chi^{\text{disc}} = 3$, which ensures that at most 10 dives are performed.

We compare the results of VRPSolver with the best results in the literature. These are the following:

**BraPed16** graph compression and arc-flow model based approach proposed in [6] and executed at a Quad-Core Intel Xeon 2.66 GHz processor,

**HesGscIrn18** stabilized branch-and-price algorithm, using a labeling algorithm for RCSP in the pricing, proposed in [19] and executed at an Intel i7-5930k 3.5GHz processor.

**WLLH20** branch-and-price algorithm proposed in [43] only for the 2-dimensional case, using a specially tailored 2-D binary knapsack algorithm for the pricing subproblem, executed at an Intel i7-6700 3.40GHz processor.

| | | **VRPSolver** | | | **BraPed16** | | **HesGscIrn18** | | **WLLH20** | |
|---|---|---|---|---|---|---|---|---|---|---|
| Class | # Items | Opt. | Time | Nodes | Opt. | Time | Opt. | Time | Opt. | Time |
| 1 | 100 | 10/10 | 29 | 1.0 | 10/10 | 67 | 10/10 | 408 | 10/10 | **1** |
| 1 | 200 | 10/10 | 164 | 1.0 | 10/10 | 7602 | 3/10 | 2899 | 10/10 | **8** |
| 2 | 200 | 10/10 | 72 | 1.0 | 10/10 | 7 | 10/10 | 1 | 10/10 | **1** |
| 4 | 100 | 10/10 | 41 | 1.0 | – | – | 10/10 | 305 | 10/10 | **1** |
| 4 | 200 | 10/10 | 270 | 1.0 | – | – | 3/10 | 2973 | 10/10 | **5** |
| 5 | 100 | 10/10 | 59 | 1.0 | – | – | 10/10 | 386 | 10/10 | **1** |
| 5 | 200 | 10/10 | 785 | 1.0 | – | – | 7/10 | 2567 | 10/10 | **1** |
| 6 | 100 | 10/10 | 28 | 1.0 | 10/10 | 1 | 10/10 | 1 | 10/10 | **1** |
| 6 | 200 | 10/10 | 163 | 1.0 | 10/10 | 5 | 10/10 | 15 | 10/10 | **1** |
| 7 | 100 | 10/10 | 17 | 1.0 | 10/10 | 2 | 10/10 | 2 | 10/10 | **1** |
| 7 | 200 | 10/10 | 113 | 1.0 | 10/10 | 14 | 10/10 | 24 | 10/10 | **1** |
| 8 | 200 | 10/10 | 18 | 1.0 | 10/10 | 1 | 10/10 | 1 | 10/10 | **1** |
| 9 | 100 | 10/10 | 36 | 1.0 | 10/10 | 28 | 10/10 | 360 | 10/10 | **1** |
| 9 | 200 | **8/10** | 961 | 44.4 | – | – | 0/10 | 3600 | 1/10 | 3541 |
| 10 | 200 | 10/10 | 140 | 1.0 | 10/10 | 155 | 7/10 | 1675 | 10/10 | **18** |

Table 4: Comparison of VRPSolver with best approaches from the literature on the vector packing instances with 2 dimensions

In Table 4 we present the results for classes of 2-dimensional instances for which the average VRPSolver solution time was more than 10 seconds. In the first column, the data class is given. Second column gives the number of items in the instances. Then for each algorithm, we show the number of instances solved within the 1 hour time limit and the average solution time in seconds. For unsolved instances, the solution time is set to the time limit.

VRPSolver clearly outperforms approaches **BraPed16** and **HesGscIrn18**. The most recent algorithm **WLLH20** is by far the fastest in almost all instances. However, VRPSolver solved the largest number of instances, including 3 open instances in class 9 with 200 items. In [43], another algorithm could solve 5 instances in class 9 with 200 items. It can be seen in Table 4 that the other classes of instances are "easy", in the sense that all their instances are solvable in root node, without need for cutting or branching.

| Class | # Items | **VRPSolver** | **BraPed16** | **HesGscIrn20** |
|---|---|---|---|---|
| 1 | 100 | **11** | 36 | 39 |
| 1 | 200 | **510** | 1374 | 2142 |
| 4 | 50 | **1278** | 3600 | 3600 |
| 4 | 100 | 3600 | 3600 | 3600 |
| 4 | 200 | 3600 | 3600 | 3600 |
| 5 | 25 | **28** | 73 | 2021 |
| 5 | 50 | 3600 | 3600 | 3600 |
| 5 | 100 | 3600 | 3600 | 3600 |
| 5 | 200 | 3600 | 3600 | 3600 |
| 9 | 200 | **131** | – | 3399 |
| 10 | 200 | 159 | **14** | 279 |

Table 5: Comparison of VRPSolver with best approaches from the literature on the vector packing instances with 20 dimensions

In Table 5 we present the results for 20-dimensional instances for which the VRPSolver solution time was more than 10 seconds. In the first column, the data class is given. Second column gives the number of items. Then for each algorithm, we give the solution time. The instance is solved to optimality if the time is less than 3600 seconds. VRPSolver is the fastest approach for the instances shown, and it solved the largest number of instances to optimality, including one open instance. All instances are solved at the root node without branching. For non-solved instances, the column generation procedure did not converge in one hour due to a large difficulty of the pricing problem.

As for the BPP, we have experimented with different orders of items. Contrary to the bin packing, item order has an impact on the solution time when solving VPP instances. However, this impact was not radical during our preliminary experiments. Moreover, we do not have any good prediction mechanism based on the instance data to decide which order is better and which one is worse. Therefore, for the final experiments, we use the same order in which items appear in original instances.

## 5.3 Variable Sized Bin Packing Problem

For the experiments on the VSBPP we use classic instances generated according to the procedure described in Monaci [23]. In addition we use instances of VSBPPOI proposed in [4]. These instances are obtained by modification of instances from [23]. There are 4 classes of instances: class 0, class 1, class 2, and class 3. In instances in class 0, all items are compulsory. These instances are generated in the same way as the original Monaci instances. In instances in classes 1 and 2, all items are optional. Instances in the first three classes contain from 25 to 500 items. In instances in class 3, there is mixture of compulsory and optional items. This class contains only instances with 500 items. All instances has been generated in [4].

The parameterisation of VRPSolver is similar to the one used for the bin packing instances except the following changes.

- When applying the path enumeration technique, the number of enumerated paths is limited to $2 \cdot 10^6$, and the number of generated labels is limited to $2 \cdot 10^5$.
- The node is finished by the MIP solver if the number of enumerated paths is less or equal to $5,000$.

We do not use initial upper bounds, but feasible solutions are obtained by the heuristic with Limited Discrepancy Search heuristic [36] with the same parameterization as for the vector packing instances. We compare the results of VRPSolver with those obtained by the branch-and-price algorithm proposed in [4], which we denote as **BCPT14**, executed at a Pentium IV 3.0 GHz processor.

In Table 6 we present the results for the instance classes 0, 1, and 2. In the first column, the data class is given. Second and third columns give the number of bin types and the number of items. Then for each algorithm, we show the number of instances solved within the 1 hour time limit, the average solution time in seconds, and the average number of nodes. For unsolved instances, the solution time is set to the time limit.

VRPSolver clearly outperforms the approach **BCPT14**, both in terms of the solution time and the number of solved instances. Note that instances in class 0 with only compulsory items seem to be solved much more efficiently by the older approaches proposed in [1] (all 300 instances solved, all average times less than 1 second) and in [17] (only two unsolved instances). We did not include their results in the Table 6 because those authors used similarly generated instances, not the original instances of [23]. Anyway, it seems that the branch-cut-and-price in [1] for the multiple length (assumes that the cost of a bin type is given by its capacity) cutting stock problem is particularly better for that kind of instances due to their *cutting stock structure*. Monaci's generation scheme creates instances with only 20 to 100 distinct weights. Solving such instances using a bin packing code like the one in **BCPT14** and in our VRPSolver model, leads to unnecessary large LPs and symmetry in the branching.

In Table 7 we present the results for the instance class 3 with 500 items. In the first column, the percentage of compulsory items. Other columns are the same as in Table 6. One can see again that VRPSolver clearly outperforms the approach **BCPT14**.

| | | | VRPSolver | | | BCPT14 | | |
|---|---|---|---|---|---|---|---|---|
| Class | # Types | # Items | Opt. | Time | Nodes | Opt. | Time | Nodes |
| 0 | 3 | 25 | 30/30 | 1 | 1.5 | 30/30 | 1 | 5.0 |
| | | 50 | 30/30 | 3 | 4.8 | 30/30 | **1** | 26.3 |
| | | 100 | **30/30** | 24 | 11.5 | 28/30 | 81 | 1190.9 |
| | | 200 | **28/30** | 268 | 43.5 | 19/30 | 1057 | 4107.8 |
| | | 500 | **28/30** | 454 | 5.5 | 13/30 | 2165 | 901.7 |
| | 5 | 25 | 30/30 | 1 | 1.0 | 30/30 | 1 | 9.9 |
| | | 50 | 30/30 | 2 | 2.3 | 30/30 | **1** | 13.1 |
| | | 100 | **30/30** | 6 | 1.4 | 29/30 | 147 | 776.5 |
| | | 200 | **30/30** | 21 | 1.3 | 22/30 | 681 | 2970.3 |
| | | 500 | **28/30** | 395 | 6.3 | 16/30 | 1908 | 1008.8 |
| 1 | 3 | 25 | 30/30 | 1 | 1.2 | 30/30 | 1 | 13.8 |
| | | 50 | 30/30 | **4** | 4.7 | 30/30 | 22 | 188.7 |
| | | 100 | **29/30** | 152 | 39.5 | 19/30 | 963 | 3297.9 |
| | | 200 | **30/30** | 366 | 90.9 | 21/30 | 1116 | 3607.3 |
| | | 500 | **23/30** | 1101 | 133.5 | 10/30 | 2561 | 1099.8 |
| | 5 | 25 | 30/30 | **1** | 1.4 | 30/30 | 9 | 100.1 |
| | | 50 | 30/30 | **3** | 5.1 | 30/30 | 46 | 429.7 |
| | | 100 | **30/30** | 57 | 45.2 | 24/30 | 626 | 1939.0 |
| | | 200 | **29/30** | 327 | 78.4 | 18/30 | 1199 | 4322.9 |
| | | 500 | **25/30** | 927 | 54.8 | 14/30 | 2054 | 933.5 |
| 2 | 3 | 25 | 30/30 | 1 | 1.1 | 30/30 | 1 | 13.2 |
| | | 50 | **30/30** | 6 | 5.9 | 28/30 | 223 | 797.3 |
| | | 100 | **30/30** | 18 | 13.5 | 22/30 | 745 | 2246.1 |
| | | 200 | **29/30** | 362 | 85.2 | 19/30 | 1209 | 4593.0 |
| | | 500 | **22/30** | 1355 | 406.9 | 11/30 | 2404 | 1030.8 |
| | 5 | 25 | 30/30 | **1** | 1.2 | 30/30 | 2 | 23.1 |
| | | 50 | **30/30** | 3 | 6.3 | 28/30 | 107 | 726.7 |
| | | 100 | **30/30** | 35 | 27.8 | 23/30 | 861 | 1974.0 |
| | | 200 | **30/30** | 194 | 43.1 | 22/30 | 1084 | 3462.6 |
| | | 500 | **19/30** | 1614 | 134.1 | 16/30 | 1960 | 836.5 |

Table 6: Comparison of VRPSolver with the approach from [4] on the variable size bin packing instances in classes 0, 1, and 2

# 6 Conclusions

This paper proposes branch-cut-and-price algorithms for the bin packing problem and for some of its well studied variants, defined as VRPSolver models. This is quite convenient, each model is coded in about 50 lines of Julia language using the package JuMP.jl [12]. Here we do not count the lines of code for reading the instance data and for solution output. The bulk of the implementation effort is the tuning of some VRPSolver parameters (important but not critical, the performance using default values is still reasonable). As far as we know, that set of models provide the most generic existing exact code for bin packing variants. Moreover, the code is freely available for academic purposes.

The computational experiments on the classic BPP indicate that:

- VRPSolver branch-cut-and-price algorithm seems to be an excellent alternative for instances that are really hard for existing exact methods, either because they are *primal-hard* or *dual-hard*. Primal-hard instances are those where an optimal solution is quite difficult to find, either by combinatorial heuristics (like [2]), by LP rounding or by diving methods. However, once an optimal solution is solution found, it is immediately proved to be optimal by Gilmore and Gomory bound. Those instances have the so-called Integer Round Up Property (IRUP). For example, AI instances are primal-hard. Dual-hard instances

| | VRPSolver | | | BCPT14 | | |
|---|---|---|---|---|---|---|
| Percentage | Opt. | Time | Nodes | Opt. | Time | Nodes |
| 0% | **6/12** | 2080 | 670.7 | 3/12 | 2820 | 1291.3 |
| 25% | **10/12** | 1033 | 412.0 | 4/12 | 2472 | 1109.0 |
| 50% | **11/12** | 945 | 28.7 | 4/12 | 2526 | 1058.5 |
| 75% | **8/12** | 1803 | 93.2 | 4/12 | 2750 | 1080.2 |
| 100% | **10/12** | 744 | 11.2 | 4/12 | 2627 | 1234.3 |

Table 7: Comparison of VRPSolver with the approach from [4] on the variable size bin packing instances in class 3

are those where finding an optimal solution is relatively easy, but Gilmore and Gomory bound is not enough to prove its optimality, cutting and/or branching is required. For example, ANI BPP instances are dual-hard. Interestingly, there are no BPP instances in the literature that are primal-and-dual-hard.

- On the other hand, on instances that are not so difficult, specialized methods may be faster, sometimes much faster. In fact, Alvim et al. [2] mention that for a significant number of BPP instances greedy heuristics (like first-fit decreasing or best-fit decreasing) find solutions that can proven to be optimal using fast lower bounding procedures (like those in [14])

A similar behaviour can be observed in the experiments with the other bin packing variants: the VRPSolver branch-cut-and-price algorithms are likely to outperform existing specialized methods on harder instances. This is explained by the fact that some advanced features in VRPSolver, like limited-memory rank-1 cuts, enumeration, hierarchical strong branching over accumulated resource consumption and limited discrepancy search diving heuristics, are more likely to make a difference on those harder instances.

As a final remark, we believe that the robust results obtained by the VRPSolver models over all those bin packing variants encourage future attempts of using that tool for solving other families of problems, not only for vehicle routing.

# Acknowledgements

Experiments presented in this paper were carried out using the PlaFRIM (Federative Platform for Research in Computer Science and Mathematics), created under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the "Programme d'Investissements d'Avenir".

The research was partially supported by the following grants: CNPq 313601/2018-6, Faperj E-26/202.887/2017, and CAPES PrInt UFF no 88881.

# References

[1] Cláudio Alves and J.M. Valério de Carvalho. A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, 35(4):1315 – 1328, 2008.

[2] Adriana CF Alvim, Celso C Ribeiro, Fred Glover, and Dario J Aloise. A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10(2):205–229, 2004.

[3] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

[4] Mauro Maria Baldi, Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Branch-and-price and beam search algorithms for the variable cost and size bin packing problem with optional items. *Annals of Operations Research*, 222(1):125–141, 2014.

[5] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85 – 106, 2006.

[6] Filipe Brandão and João Pedro Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56 – 67, 2016.

[7] Alberto Caprara and Paolo Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111(3):231 – 262, 2001.

[8] Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129 – 146, 2014.

[9] Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019.

[10] Maxence Delorme and Manuel Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32(1):101–119, 2020.

[11] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[12] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[13] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[14] Sándor P Fekete and Jörg Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical programming*, 91(1):11–31, 2001.

[15] Sylvie Gélinas, Martin Desrochers, Jacques Desrosiers, and Marius M. Solomon. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61(1):91–109, 1995.

[16] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[17] Mohamed Haouari and Mehdi Serairi. Relaxations and exact solution of the variable sized bin packing problem. *Computational Optimization and Applications*, 48(2):345–368, 2011.

[18] Stephan Held, William Cook, and Edward C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.

[19] Katrin Hessler, Timo Gschwind, and Stefan Irnich. Stabilized branch-and-price algorithms for vector packing problems. *European Journal of Operational Research*, 271(2):401 – 419, 2018.

[20] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[21] Mads Jepsen, Bjorn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

[22] Leonid V Kantorovich. Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422, 1960. Translation of a 1939 paper in Russian.

[23] Michele Monaci. *Algorithms for packing and scheduling problems*. PhD thesis, University of Bologna, 2002.

[24] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. In *Proceedings of the XVII IPCO*, volume 8494 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 2014.

[25] Diego Pecin, Claudio Contardo, Guy Desaulniers, and Eduardo Uchoa. New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 29(3):489–502, 2017.

[26] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[27] Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos. Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters*, 45(3):206 – 209, 2017.

[28] Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. Enhanced branch-cut-and-price algorithm for heterogeneous fleet vehicle routing problems. *European Journal of Operational Research*, 270:530–543, 2018.

[29] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

[30] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. Technical Report L-2019-2, Cadernos do LOGIS-UFF, Niterói, Brazil, June 2019.

[31] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 354–369. Springer, 2019.

[32] U Pferschy, H Kellerer, and D Pisinger. Knapsack problems, 2004.

[33] David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

[34] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland, 1981.

[35] Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. A bucket graph based labeling algorithm with application to vehicle routing. *Transportation Science*, accepted, 2020.

[36] Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri, and Eduardo Uchoa. Primal heuristics for branch-and-price: the assets of diving methods. *INFORMS Journal on Computing*, 31(2):251–267, 2019.

[37] J. E. Schoenfield. Fast, exact solution of open bin packing problems without linear programming. Technical report, US Army Space and Missile Defense Command, 2002.

[38] José M Valério de Carvalho. Using extra dual cuts to accelerate column generation. *INFORMS Journal on Computing*, 17(2):175–182, 2005.

[39] Pamela H Vance, Cynthia Barnhart, Ellis L Johnson, and George L Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational optimization and applications*, 3(2):111–130, 1994.

[40] François Vanderbeck and Laurence A Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

[41] Gerhard Wäscher and Thomas Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *Operations-Research-Spektrum*, 18(3):131–144, 1996.

[42] Laguna Wei, Zhixing Luo, Roberto Baldacci, and Andrew Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, Online First, 2020.

[43] Lijun Wei, Minghui Lai, Andrew Lim, and Qian Hu. A branch-and-price algorithm for the two-dimensional vector packing problem. *European Journal of Operational Research*, 281(1):25 – 35, 2020.